NRL Memorandum Report 5570

A Vectorized "Nearest-Neighbors" Algorithm of Order N Using a Monotonic Logical Grid

J. P. Boris

Laboratory for Computational Physics

May 29, 1985

This work was supported by the Office of Naval Research.

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Memorandum Report 5570	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Research Laboratory — 6b OFFICE SYMBOL (If applicable) 4040	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Washington, DC 20375-5000	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING / SPONSORING ORGANIZATION Office of Naval Research — 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) Arlington, VA 22217	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
RR-011-09-43			DN280-069

11 TITLE (Include Security Classification)

A Vectorized "Nearest-Neighbors" Algorithm of Order N Using a Monotonic Logical Grid

12 PERSONAL AUTHOR(S)
Boris, J.P.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Interim	FROM ______ TO ______	1985 May 29	50

16 SUPPLEMENTARY NOTATION

This work was supported by the Office of Naval Research.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

When a large number of separate objects interact, N*(N-1)/2 interactions can occur. At any instant a given object may interact strongly with only a few of the N - 1 others. Unfortunately, keeping lists of the other objects with which it interacts or recomputing these nearest neighbors each timestep is computationally expensive. These "nearest-neighbors" problem has persisted in computational physics and computational geometry for several decades. We need efficient algorithms which select important nearest-neighbor interactions without having to check and analyze N^2 interactions. To date the best algorithms which scale as N, rather than N^2, are scalar algorithms which address memory randomly.

This report introduces an efficient 3D nearest-neighbors algorithm whose cost scales as N and which vectorizes easily using data from contiguous memory locations. A Monotonic Logical Grid (MLG) for storing the object data is defined dynamically so that objects which are adjacent in real space automatically have close

(Continues)

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL J. P. Boris	22b TELEPHONE (Include Area Code) (202) 767-3055 — 22c OFFICE SYMBOL Code 4040

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT (Continued)

address indices in the compact MLG data arrays. The data values for each object are stored at a location (i,j,k) in the MLG such that the X positions of all the objects increase monotonically with index i, the Y positions increase monotonically with index j, and the Z positions increase monotonically with index k. Such a well-structured mapping from the real positions to regular, compact data arrays can always be found. Further, when object motions result in a local violation of spatial monotonicity, another MLG always can be found nearby. This means that local changes in the object positions and hence spatial ordering do not trigger global changes in where these object data are stored in the MLG.

The data relocations required to maintain the MLG as objects pass each other in space can also be vectorized efficiently. The MLG algorithms will execute effectively in small array processors and partition to take advantage of asynchronous parallel architectures in VLSI/VHSIC-based supercomputer systems of the future. Using a commercially available distributed processing system, 5000 interacting objects could be monitored and the MLG data base updated and restuctured thousands of times in about 15 minutes, fast enough to be useful for real time applications as well as physics simulations.

CONTENTS

A VECTORIZED "NEAREST-NEIGHBORS" ALGORITHM OF ORDER N USING A MONOTONIC LOGICAL GRID

SECTION I. INTRODUCTION AND BACKGROUND

When N independent objects interact in space, N(N-1)/2 interactions might be important in determining how a given object reacts to the others at any instant. Usually exact positions and velocities of the neighboring objects must be known. Knowing statistical averages and the general properties of the distribution of objects nearby does not provide enough data to compute local interactions accurately. At any instant a given object may interact strongly with only a few of the others. Unfortunately, keeping track of or repeatedly recomputing which ones are near neighbors is computationally expensive. The goal is efficient, simple algorithms which select the nearest-neighbors without a computational premium scaling as N^2. Effort on the nearest-neighbors problem has persisted in computational physics and computational geometry for several decades. This report introduces an efficient 3D nearest-neighbors algorithm whose cost scales as N and which vectorizes easily using contiguous memory locations.

An efficient vector solution of the nearest-neighbors problem would advance many important applications. For an important class of molecular dynamics problems involving interactions among many atoms and molecules, the nearest neighbors exert the strongest forces and are the most likely candidates to enter into chemical reactions. Many important physics problems in gases, liquids, solids, and transitions among these phases require detailed manybody calculations where the close encounters are most important.

Manuscript approved February 19, 1985.

For graphics based on vertex-edge representations of complex 3D shapes, local relationships and orientations of nearby vertices determine which surfaces are visible. It is clearly advantageous to be able to construct a 2D image of a complex 3D scene, for example, using the parallelism made possible by Very Large-Scale Integration (VLSI). Terrain management simulation models and multi-dimensional radiation transport models are currently limited in their ability to compute geometric obscuration. For controlling airline traffic over crowded airports, collisions with nearby planes are the most immediate danger - and demand shorter timescales for detection and corrective response. Consider a related scenario for futuristic battle area management. A one-pass engagement against many thousands of high-speed opponents requires fast redetermination of nearest neighbors to ensure effective retargeting in real time. These applications all require rapidly updating many distinct local configurations as the objects move.

For complex manybody problems with N = 5000 independent objects, more than thirty thousand of degrees of freedom are required, and 12.5 million interactions exist which ideally ought to be considered. Current supercomputers deliver ~50 Megaflops (million floating point operations per second) on optimized but realistic problems. The straightforward recalculation of all interactions requires about 60 vectorizeable operations per interaction, or 10 - 15 seconds of dedicated supercomputer time. This is not fast enough for real time applications where the data base should be updated and the neighbors recalculated every second or two.

This report introduces a simple three-dimensional nearest-neighbors algorithm whose cost scales as N, not as the square of N, and which vectorizes easily using data from contiguous memory locations. A compact data structure to store the object data, called a Monotonic Logical Grid (MLG), is defined dynamically so that objects which are adjacent in real space automatically have close address indices in the MLG data arrays as well. As two objects move past each other in space, their data are exchanged or "swapped" in the MLG data arrays to keep a strictly monotone mapping between the geometric locations and the corresponding storage location indices.

To construct an MLG the data values for each object are stored at location (i,j,k) in the MLG such that the X positions of all the objects increase monotonically with index i, the Y positions increase monotonically with index j, and the Z positions increase monotonically with index k. Section III describes the algorithm in some detail.

It is not obvious but it is true that such an organized logical ordering of even random locations can always be found. In Section III an order N log N constructive algorithm for one such MLG is provided proving existence. Generally more than one MLG meeting all the monotonicity conditions seems to be possible so the technical problem of selecting the optimum MLG for a particular application has to be addressed. In one case, minimizing average distances to neighbors in the MLG may give the best grid. In other problems it may be best to maximize the shortest distance to any point which is not a near neighbor in the logical grid.

Further, when object motions result in a local violation of the monotonicity conditions on which the original MLG was based, another MLG can be found nearby. This means that local changes in the object positions and hence spatial ordering do not trigger global changes in where these object data have to be stored in the MLG. The data relocations to maintain the MLG as objects pass each other in space can be vectorized without inefficient gather/scatter operations or variable-length (scalar) linked lists. The MLG data structure and algorithms allow contiguous-data vector operations which are long enough to be efficient for physical force sums, for $F = Ma$ orbit integrations, and for the object data "swapping" used to restructure the MLG whenever the monotonicity conditions are violated.

The cost to execute a simple test version of the model is one hour on a DEC VAX 11/780 for one thousand particles for 1000 timesteps. We used a power series force law for the 124 nearest neighbors, assuming that the average particle separation distance is smaller than the cutoff radius R_c of the force law. A commercial distributed processing system, a minicomputer host with modest array processors, would be fast enough using an MLG to integrate 5000 interacting objects and restructure the data base thousands of times in about 15 minutes, useful for realtime applications where current supercomputers using other algorithms will be inadequate. The MLG also permits partitioning to take advantage of asynchronous multiprocessor parallelism in VLSI/VHSIC-based distributed processing systems (e.g. Fox and Otto, 1984).

The computational costs of several algorithms and combinations of algorithms which have been proposed to reduce the cost of finding nearest neighbors are discussed in Section II. Section III contains a description of the MLG itself, an O(N log N) sort algorithm to find a starting MLG from arbitrary initial data, and simple algorithms which restructure the grid dynamically as the objects move. Section III also presents a few simple tests of the method. Section IV considers several extensions. Section V contains a summary and conclusions.

SECTION II. THE COMPUTATIONAL COST OF FINDING NEAREST NEIGHBORS

As described in Section I, brute force recalculation of all the interactions can be vectorized but is of order N^2 and therefore costly. The best nearest-neighbors algorithms published, Hockney and Eastwood (1981), are of order N with minimal operation counts. However, these O(N) algorithms are intrinsically scalar and execute relatively poorly in parallel or pipeline-architectured supercomputers. Further, memory is addressed essentially at random so data buffering from disk or virtual memory for a large problem is time consuming.

In the next few paragraphs the computational costs of simple strategies to reduce the nearest-neighbors problem are compared. The assumed operation counts are only representative values since optimal implementations are always machine and problem dependent (Gunsteren, et al., 1984). There are too many possible algorithms and variations to compare all of their operation counts. It is even harder to compare scalar and vector algorithms, something we would like to do in theory here but which can really only be done in practice. As a rule of thumb, efficient use of the vector hardware in supercomputers or array processors generally produces over an order of magnitude speed up over reasonably well compiled scalar code. In some cases the vector speed-up factor will be greater and in others, less.

For problems where the number of near neighbors is large so the computational cost is dominated by the physical interaction calculations rather than computational book-keeping to find the near neighbors, the algorithm introduced here calculates two to three times as many interactions as minimally necessary. This is accepted as the price for simple logic and vectorized computation in contiguous memory. This means that a computer whose vector speed is only a few times the scalar speed may see no improvement over the Hockney-Eastwood PPPM techniques. In computers where the vector-scalar ratio is large, an order of magnitude improvement with an MLG is at least conceivable. More substantial gains are possible in highly parallel multi-processor systems because the MLG algorithms partition naturally.

Let N_{ot} = ~60 be the total number of floating point operations (flops) used to evaluate each interaction between two of the N = 5000 objects. The main component of the cost for a timestep will be

$$\begin{aligned} \text{\# Flops to compute all interactions} &= \text{Fcai} \\ &= N \times (N \times N_{ot})/2 \qquad (2.1) \\ &= 7.5 \times 10^8 \text{ flops} \rightarrow 15 \text{ seconds/step at 50 Megaflops.} \end{aligned}$$

Manybody calculations which compute all interactions have become prohibitively expensive with even a few hundred objects because thousands of timesteps are required for complex problems. The operation count per timestep goes up quadratically with the number of particles N but the effective resolution only increases as the cube root of the number of particles. This scaling of cost with at least the sixth power of resolution is prohibitive. If the number of timesteps also has to be increased when more objects are simulated, the scaling can be even worse. This brute force $O(N^2)$ algorithm is of interest because it vectorizes and partitions easily and is exceedingly simple.

Reduction of this computational expense is obtained by computing the details of the interactions only for pairs of objects closer than a cutoff distance R_c. This basic nearest-neighbors concept takes its most sophisticated form in the "Particle-Particle-Particle-Mesh (PPPM) algorithms of Hockney and Eastwood (1981). Faster algorithms and data structures for implementing this nearest-neighbor approximation have been the subject of much computational research in the last few decades.

Checking two locations to see if they are within a distance R_c of each other requires about $N_{oc} = {}^{\sim}10$ floating point operations. Nine or ten flops are required simply to calculate the square of the distance between the two objects and then compare it with the square of R_c.

To calculate the physical forces and sum them for each interaction pair separated by less than R_c requires

$$N_{oi} = N_{ot} - N_{oc} = {}^{\sim}50 \text{ floating point operations} \qquad (2.2)$$

per interaction. This would be enough operations to calculate, for example, a simple force law with transcendental functions or to estimate a collision cross-section. If $N_{nn} = {}^{\sim}60$ neighbors must be considered for each object, our formula for the number of flops to compute near interactions is

$$\begin{aligned} \# \text{ Flops to compute near interations } &= \text{Fcni} \\ &= N \times (N/2 \times N_{oc} + N_{nn} \times N_{oi}) \qquad (2.3) \\ &= 1.4 \times 10^8 \text{ flops} \rightarrow 2.8 \text{ seconds/step at 50 Megaflops.} \end{aligned}$$

Using the concept of a cutoff radius R_c has reduced the operation count by over a factor of five. Just checking all the interaction distances requires appreciable time when $N = 5000$ and N_{oc} is a modest fraction of N_{oi}. The computational cost still scales as N^2 and the factor of five speedup shown above is largely illusory as it ignores the cost of organizing the neighboring object data into contiguous vectors for efficient computation of the physical interactions. If this were not done, the interaction calculations would have to be performed as scalar operations. It would be worth organizing these vectors if enough objects interact with each other on average but this further optimi-zation would be wasted if the average particle separation distance were bigger than R_c.

In practice the N_{nn} near neighbor variables have to be gathered into vectors, costing about N_{gv} = ~ 50 flops per neighbor per object. The vector lengths are also shorter, engendering some additional overhead. Thus a more accurate estimate of F_{cni} is

$$\begin{aligned} \#\ \text{Flops to compute near interactions} &= F_{cni} \\ &= N \times (N/2 \times N_{oc} + N_{nn} \times (N_{gv} + N_{oi})) \qquad (2.4) \\ &= 1.75 \times 10^3 \text{ flops} \rightarrow 3.5 \text{ seconds/step at 50 megaflops.} \end{aligned}$$

Reducing the number of nearest neighbors used does not help much as long as cutoff radius checking is being done for all possible inter-actions. In fact, as long as this is done, it hardly hurts to double the number of neighbors kept for calculating the interactions. Clearly, too much time is spent checking interaction distances which are too large to be significant.

It is natural to consider updating the lists of near neighbors less often. If the $O(N^2)$ selection of near neighbors is only done every N_{sc} = ~2.5 cycles, the cost for this portion of the algorithm is reduced. However, there are additional costs and N_{sc} cannot be very large. The number of near neighbors has to be augmented to allow for motion of particles near R_c into and out of the cutoff region. This increase is roughly the same as the relative volume change due to particles moving into and out of the sphere of radius R_c during the N_{sc} steps between recomputation of the near neighbors lists. Let an average particle travel a small fraction f of R_c per timestep. The sphere of near neighbors will increase in volume by a factor of $\sim(1 + f \times N_{sc})^3$ in the N_{sc} steps until the neighbor lists are updated. When the sphere has roughly doubled in volume, a radial change of 25%, the lists must be updated since most of the neighbors will have changed. Thus, with f = 0.1

$$N_{sc} = .25/f = \sim 2.5 \qquad (2.5)$$

and N_{nn} must be multiplied by $(1 + f \times N_{sc})^3 = \sim 2$ in the above formula to give

$$\begin{aligned} \#\text{ Flops to intermittently recalculate neighbors} &= F_{irn} \\ &= N \times (N/2 \times N_{oc}/N_{sc} + N_{nn} \times 2 \times (N_{gv} + N_{oi})) \qquad (2.6) \\ &= 1.1 \times 10^8 \text{ flops} \rightarrow 2.2 \text{ seconds/step at 50 megaflops.} \end{aligned}$$

Although the N^2 search is a factor of 2.5 less important, counteracting effects eat up some of this gain. If f were reduced to 0.01, probably reasonable for molecular dynamics simulations, some additional gains could be realized. However, these would be better by less than a factor of two because the cost is now in computing the near interactions as it should be. In other applications, however, there would not necessarily be a need to use the short timesteps required by physical simulations such as molecular dynamics.

The only way to avoid the N^2 premium is to update the nearest-neighbors list of each object using objects in a volume larger than would be required for an interaction cutoff of R_c but much smaller than the entire system. Hockney and Eastwood define a PPPM "chaining mesh" where $dX = dY = dZ = R_c$ and check distances to objects known to be in only the nearest 13 $= (3^3 - 1)/2$ cells. Only these particles might be within R_c of a particle in the chaining cell under consideration. On average only about 40% of the particles in these 13 cells are actually within R_c. Taking L as a typical system dimension, there are $N_{cc} = N\,(R_c/L)^3$ particles in each of the PPPM cells. The number of cutoff distance checks performed in a timestep is then

$$\#\ \text{PPPM checks} = N \times 13 \times N_{cc}. \qquad (2.7)$$

In the PPPM formulation, when R_c is twice the average spacing, a typical particle has its distance to 104 other nearby particles checked (13 cells × 8 particles per cell). The corresponding number in the MLG would be 62 if all interactions within two grid displacements in any direction are kept. This nominal factor of two gain in the MLG approach is lost again because all the interactions would be calculated to maintain vectorization rather than only 40% as possible with the scalar PPPM algorithm. The real gain is the ability to use efficient, contiguous memory, vector operations throughout the MLG algorithms and to cleanly partition the problem into computational subtasks.

The operation count for the overall MLG algorithm developed next in Section III is also problem dependent. Using typical simulation variables summarized in Table 2.1 below, the cost of the MLG in vector floating-point operations to execute a timestep, exclusive of the relatively inexpensive orbit calculations, is

$$\begin{aligned}
\#\ \text{Flops for the Monotonic Logical Grid algorithm} &= F_{mlg} \\
&= N \times (N_{nn} \times N_{ot} \quad \text{for neighboring object interactions} \\
&\quad + 3 \times N_{si} \times N_{os}) \quad \text{for swapping iterations in X, Y, Z} \\
&= 2.25 \times 10^{7}\ \text{flops} \rightarrow \sim 0.5\ \text{seconds/step at 50 megaflops.}
\end{aligned} \qquad (2.8)$$

Here $N_{si} = \sim 4$ is the number of iterations of vector swapping performed over the entire grid to restructure the MLG after the object positions change each timestep. $N_{os} = \sim 60$ is the number of floating point operations to execute a single swap of two objects in the MLG. In the next section the concepts and details of the Monotonic Logical Grid Algorithm are presented.

Table 2.1. Typical Values of Manybody Simulation Variables

N	=	5000	=	# of objects interacting in space
N_{os}	=	~60	=	# operations per vector swap in MLG algorithms
N_{si}	=	≤ 4	=	average # of vectorized swapping iterations to relocate object data in the MLG
N_{nn}	=	~60	=	# of near neighbors usually included in the interaction calculations
N_{oc}	=	~10	=	# of flops to check distance between two objects
N_{oi}	=	~50	=	# of flops to interact two objects
N_{ot}	=	~60	=	# of flops total to compute an interaction
N_{gv}	=	~50	=	equivalent # of arithmetic flops to gather neighboring object data from random locations in memory
N_{sc}	=	~2.5	=	# of steps between recomputation of the nearest-neighbors lists in scalar algorithms
N_{cc}	=			# of objects in the average cell of PPPM chaining mesh.

III. THE MONOTONIC LOGICAL GRID ALGORITHM

A Monotonic Logical Grid (MLG) is a simple, compact way of indexing and storing the data describing a number of objects moving in space. For N particles in three dimensions, the arrays of object locations, X(i,j,k), Y(i,j,k), and Z(i,j,k), constitute an MLG if and only if

$$\begin{aligned} X(i,j,k) &\leq X(i+1,j,k) \quad \text{for} \quad 1 \leq i \leq NX-1, \\ Y(i,j,k) &\leq Y(i,j+1,k) \quad \text{for} \quad 1 \leq j \leq NY-1, \text{ and} \\ Z(i,j,k) &\leq Z(i,j,k+1) \quad \text{for} \quad 1 \leq k \leq NZ-1. \end{aligned} \tag{3.1}$$

Given N = NX*NY*NZ random locations, the spatial lattice defined by an MLG is irregular. However, the cells defined by logically neighboring locations are distorted cubes and thus form a useful consistent partitioning of the spatial volume. When the N object locations satisfy Eqs. (3.1) and any additional constraints or relations specifying other than infinite-space boundary conditions, they are in "MLG order". This ordering is useful because the direction for going from one object to another in space and in the MLG are the same. Further, other objects which are between two objects in space will also be between them in the MLG. Thus neighbors in real space have neighboring address indices in the MLG as well.

Figure 3.1 shows three different spatial configurations of 16 objects in the lower three panels. The objects are ordered into four rows and four columns in each of these configurations corresponding to regular storage of the object data in the two-dimensional MLG shown above. The cells of the MLG move with the objects and thus always have exactly one object in them. When all the objects move to the upper left of the region, as in the lower right panel, the MLG is just as regular as when the objects are uniformly spaced. This mapping of irregular locations onto a very regular data structure is what permits optimal use of vector and multiprocessor hardware.

Figure 3.2 illustrates several different MLG mappings of the same 16 object locations. The upper left panel shows the 16 locations in a regular spatial lattice. The obvious numbering of the locations into four rows of four objects each is an MLG because the X and Y components of all the object locations (dots) increase monotonically with the X and Y indices, i and j. In the remaining three lower panels the locations have been displaced from the regular spacing. Each of these panels contains the same data, but the MLGs for storing these data, as indicated by the logical mesh lines, all differ. The lower left panel is a recognizeable distortion of the regular grid above. The indexing of the objects would be identical in both cases although the actual data stored would differ somewhat since the objects have moved away from their regular locations. The lower center and lower right panels show different logical indexing for the same physical data giving two other Monotonic Logical Grids (MLGs). In the center the connections to points in the second row from above and from below have all been displaced to the left. On the right, the connections to the second column from objects located logically in columns one and three have been displaced downward.

These figures show that there can be a number of MLGs with the same Lagrangian object data, all satisfying the required monotonicity conditions from Eqs. (3.1). These spatial monotonicity conditions constitute $3N - NX\times NY - NY\times NZ - NZ\times NX$ numerical comparisons which can be performed to determine if a particular organization of the object locations is in MLG order. For each dimension of the desired data structure such a monotonicity condition can be defined.

In space the coordinates can be rotated or redefined and this corresponds to a different family of MLGs. The monotonicity conditions may not change in the new coordinate system but the object location coordinates will. Even if the coordinate system is held fixed, the object motions will quickly invalidate at least some of the relations (3.1) requiring a reorganization of the object data in the arrays to store a completely monotone mapping. Using the monotonicity conditions, a given data structure can be checked efficiently to see if the locations are in MLG order. However, additional algorithms are needed when MLG order is violated.

If the objects are not in MLG order, the following algorithm using a vector sort routine O(N log N) can be used to rearrange them. First sort all N locations into the order of increasing Z. The first NX x NY of them, should be indexed k = 1, and sorted into the order of increasing Y. The first NX of these, should be indexed j = 1 and then sorted into the order of increasing X. These objects are indexed from i = 1 to i = NX. The next NX locations, indexed j = 2 but still k = 1, are again ordered and indexed from i = 1 to i = NX. This procedure is continued until the first NX x NY plane of locations has been arranged. Since the locations were initially ordered in Z, the subsequent reorderings within the k = 1 plane cannot disturb the monotonicity conditions relating the first plane to any subsequent reordering of the second and subsequent planes. Similarly, all the locations will satisfy the monotonicity conditions in Y and X as well.

Once the first plane is ordered, the next NX x NY locations are indexed k = 2, and the MLG ordering within this plane is constructed just as for the first plane. All NZ planes are organized this way. The process requires of order

$$\begin{aligned} & NZ \times NY \times NX \times (\log NZ + \log NY + \log NX) \\ & \quad + NZ \times (NY \times NX \times (\log NY + \log NX) + NY \times NX \times \log NX) \qquad (3.2) \\ & \quad = NZ \times NY \times NX \times (\log NZ + 2 \log NY + 3 \log NX) \end{aligned}$$

operations to construct the MLG. This sort algorithm could be repeated every timestep as necessary to restructure the MLG when object motions in one of the three coordinate directions cause some of the conditions (3.1) to be violated.

The existence of this constructive algorithm proves that at least one MLG for even random locations always exists and that it is not hard to find. As a consequence, data manipulation and summation algorithms in the MLG can always assume the rigorous spatial monotonicity of the MLG. When several object locations are identical, any ordering the sort procedure comes up with is correct as the conditions (3.1) are satisfied. Locally degenerate grids are possible when several locations overlap.

Although this algorithm is fast, it has two limitations: it is of order N log N, not N, and it may move data a long distance in index space to correct even small changes in position. To counter these objections, an order N algorithm is described which executes local but vectorizeable exchange or "swapping" operations on the MLG data to restore monotonicity everywhere. The extra factor log N is removed because small monotonicity upsets from the previous set of locations generally do not require information from the other side of the grid for their correction.

If two objects move less than a typical separation distance per timestep, a condition generally required for accurate integration of the equations of motion, a few iterations are usually enough to restore MLG order. A "swap" is executed by testing the conditions in Eqs. (3.1), and then, when the corresponding monotonicity condition is violated, exchanging the locations in the logical grid of all data pertaining to the two objects involved. Each direction is checked separately. A red-black algorithm (Adams and Jordan, 1984) would allow at least half the tests in a given direction to be performed simultaneously and thus vectorized while converging as fast as a scalar iteration.

No more than five arithmetic operations are required to test for monotonicity and to prepare to swap any amount of data. A floating point 0.5 is logically "or"ed with the sign bit of the coordinate difference to obtain a number S which is 0.5 if the two coordinates are in MLG order and -0.5 if they are not. This takes three operations. Two more operations give the weights w and (1 - w) where

$$w = S + 0.5, \qquad (1 - w) = S - 0.5. \tag{3.3}$$

The weight w is unity when the coordinates are in order and zero when they are not. If the test $X(i,j,k) \leq X(i+1,j,k)$ is being performed, the X components of the object locations can be exchanged using the following formulae (six operations):

$$\begin{aligned} T(i,j,k) &= w \times X(i,j,k), \\ U(i,j,k) &= (1 - w) \times X(i,j,k), \\ X(i,j,k) &= T(i,j,k) + (1 - w) \times X(i+1,j,k), \text{ and} \\ X(i+1,j,k) &= w \times X(i+1,j,k) + U(i,j,k). \end{aligned} \tag{3.4}$$

When the locations are in MLG order, the swapping formulae change nothing. When two locations are out of order, these formulae interchange the object data on the MLG so they will be in order for the next iteration. All object data at every grid point can be treated identically with this procedure.

The algorithm vectorizes easily even though some of the pairs of objects have to trade places in the MLG and others do not.

These six operations must be repeated to swap each data variable stored in the MLG. As a minimum these include the three components of the object locations and an identification number, ID#(i,j,k), to mark which of the N particles currently is at i,j,k in the MLG. To vectorize the complete algorithm, the velocity components VX(i,j,k), VY(i,j,k), VZ(i,j,k), the mass M(i,j,k), and another force law constant FC(i,j,k) must also be moved about dynamically. These nine variables require 54 operations to be moved between adjacent cells for each swapping iteration. Thus Nos = ~60 operations are required for each iteration in each direction for each object. This is about as much work as calculating three components of the force acting between two objects which are near neighbors in the MLG. With Nsi = ~4 swapping iterations being performed in each direction, the total cost of restoring the MLG every timestep is about the same as calculating forces from 12 neighbors. When timesteps are short, this can be reduced even further.

When the MLG algorithm is used, the cost in vector floating-point operations to execute the geometric and force summing in a timestep is given by Eq. (2.8). The speedup expected using this algorithm is large, a factor greater than thirty for 5000 objects. Not only is the N^2 dependence removed but the actual nearest-neighbor interactions can be computed with very high efficiency, comparable to the best order-N scalar algorithms. Only about a fifth of the computation is expended on maintaining the MLG data structure. The rest is used in computing pairs of interactions at full vector efficiency.

The random motion of point particles in a cubical domain is taken as a test problem to illustrate the concepts. A topologically regular 8 × 8 × 8 3D grid is defined for storing the position and velocity components of 512 randomly located objects. The domain is doubly periodic in X and Y and is bounded in Z by two reflecting end walls at Z = 0 cm and $Z = Z_{max} = 8$ dZ. A number of short calculations have been performed using this system to test and develop various aspects of the model. Figure 3.3 shows the first of eight planes of this 3D MLG, plotting the X and Y locations of the 64 objects currently on that plane. The initial conditions for the calculation are shown in the upper right, regularly-spaced locations with random velocities uniformly distributed in each coordinate from -10^7 cm/sec to $+10^7$ cm/sec. The three remaining panels show plots of the 64 locations in the same MLG data plane at three times. As the objects move in the plane and between planes, a complicated but clearly structured MLG is always maintained.

Under a number of different physical circumstances and numerous different initial conditions the model has been able to find an MLG after only a few swapping iterations. The average near-neighbor separations increase somewhat at first over their almost minimal initial values. Rather quickly, however, random swapping halts the increase of this average distance to the near neighbors. Figure 3.4 displays the frequency distribution for the number of swapping iterations required to restore the MLG after relative motion of the objects has disrupted it. Three cases were run from the same physical initial conditions and zero-sized non-colliding particles, with timesteps $\delta t = 2.5 \times 10^{-16}$ sec, 1.0×10^{-15} sec, and

4.0×10^{-15} sec. The lightly shaded bars in Figure 3.4 correspond to the intermediate case with 10^{-15} sec as the timestep. For this case $dl_{max} = .1\Delta$ meaning that the fastest particle traverse 1/10 of the regular initial spacing of $\Delta = 10^{-7}$ cm per timestep. The data with unshaded bars, $dl_{max} = .025\ \Delta$, shows the results when δt is smaller by a factor of four and the data depicted with dark bars shows results when δt is a factor of four larger, i.e. $dl_{max} = .4\Delta$.

To interpret the figure consider $dl_{max} = 0.1\Delta$. About 40% of the timesteps (frequency 0.38) required 4 iterations of swapping to restore the MLG. Less than 10% of the timesteps required 6 or more iterations. The average number of iterations required is 4.0 for $dl_{max} = 0.1\Delta$. When $dl_{max} = 0.025\Delta$, the average number of swapping iterations is 2.85, about 2/2. When $dl_{max} = 0.4\ \Delta$, the average is 5.0 swapping iterations per timestep. Thus the actual computational work decreases per unit integration time with longer timesteps because the number of swapping iterations increases much more slowly than the timestep increases.

A great deal of swapping goes on in the first few iterations out to the average number for the particular timestep chosen. For timesteps with relatively large numbers of iterations I, the likelihood of this extra work being required decreases by a factor of two or three for each extra iteration. These timesteps requiring a relatively large amount of work contribute very little to the average computation load needed to restore the MLG because they occur so infrequently.

In test calculations, with nonzero particle size, forces were calculated between a given object and the 5 × 5 × 5 cubical nearest neighbors interaction template of 125 neighboring objects centered on it in the MLG. Since the interaction has to be computed only once for a pair of objects and can be ignored for self interactions, the tests had the following number of near neighbors

$$N_{nn} = (5 \times 5 \times 5 - 1)/2 = 62 \approx 60. \qquad (3.6)$$

When many objects are within the cutoff distance R_c, the interaction template should be extended, perhaps to 7 × 7 × 7. An appreciable fraction of the forces calculated will be beyond the cutoff distance but this extra work is compensated by the fact that all the work can now be performed by vec. r operations working from contiguous locations in the computer storage. This gain is typically an order of magnitude or more in speed and is still worthwhile even if a factor of two or three is wasted calculating unnecessary interactions.

When objects are far apart compared to the cutoff radius R_c, only the 13 neighbor interactions from the 3 × 3 × 3 interaction template need be considered. This number 13 is the same as the number of chaining cells which have to be considered in Hockney's PPPM data structure to find all objects within the cutoff radius R_c. Figure 3.5 shows a schematic rendition of these different interaction templates. Only the half of the template with index offset larger than zero has to be considered since all interactions with objects having a lower storage address index will have been calculated previously. As shown, shells of interaction can be defined which will correspond approximately to neighbors at different physical distances. The 16 neighboring nodes indicated with grey squares form the closest shell. The 30 triangle nodes are a bit further away, on average, and the 16 circle nodes form the furthest shell of the nearest neighbors template.

SECTION IV. ADDITIONAL ASPECTS OF MONOTONIC LOGICAL GRIDS

IV.A Possibilities for Further Optimization

By a fully vectorizeable process of exchanging or "swapping" objects between adjacent logical cells every few timesteps, the nearest-neighbor MLG ordering is kept intact even though the objects move from cell to cell. Thus the method can be applied to gas, solid, and liquid systems using the same logical structure for problems of interesting size, i.e. 1,000 - 10,000 particles. The MLG algorithms forego a regular grid in space with a variable number of objects in each cell for an irregular spatial grid which has exactly one object per cell by construction. This logical simplification, brought about by the MLG mapping, permits extensive optimization under current and planned supercomputer architectures (e.g. Fox and Otto, 1984) without sacrificing the generality needed to make it useful.

Optimization of nearest-neighbor algorithms for particle dynamics is both machine and problem dependent. Vectorization techniques to achieve very high rates of computation require that all logical and arithmetic operations be performed on organized arrays of independent data. Distributed processing approaches to massive parallelism rely on a number of self-controlled processing centers operating asynchronously, but according to fixed rules of cooperation, on an evolving data base. To take advantage of both approaches simultaneously requires being able to define a number of vectorizeable segments of the problem which can be calculated independently. Furthermore, the vectors must be long enough to be computationally efficient but short enough that the memory needed in each asynchronous processing center is not prohibitively expensive. The MLG algorithms presented in the previous section can be partitioned for multi-tasking across a number of independent processors.

To maximize the length of vectors within each partition when the typical MLG dimension, $NX \approx NY \approx NZ \approx N^{1/3}$, is only about 20 (8,000 objects) requires treating a substantial fraction of a plane as a single vector. In the $8 \times 8 \times 8$ test problem, vectors of length 64 can be used throughout except for the X-direction monotonicity tests where vectors half as long would result. This is accomplished by collapsing several indices into one index and by paying careful attention to the boundary conditions.

Optimum computational efficiency results when the last few swapping operations are performed only for the grid points which might have become non-monotone due to adjacent swaps taking place during the previous iteration. Reductions of up to a factor of two in computer work to maintain monotonicity might be obtained by reducing the number of inactive vector swap attempts greatly at the cost of considerably increased program complexity. The scalar program to perform the few remaining swaps and keep track of which few nodes might have had their monotonicity conditions affected by the previous swaps is complicated. To date, convergence of the number of vector swapping iterations required has been so fast that this extra work has not been indicated. In the future it may be worth the effort for production calculations.

The same kind of gain can be obtained by trimming the near neighbors template defining which logical neighbors are likely enough to be close spatial neighbors that they should be included in the vector interaction calculations automatically. When a scalar "clean-up" portion is added to the vector force summing algorithm, the number of logically neighboring nodes which are always considered can be reduced significantly below that required to ensure no close uncounters. Figure 3.5 shows three shells of logical interactions in the nearest neighbors template, each succeeding shell taking neighbors which are logically, and usually physically, farther away.

By keeping track of maximum X, Y, and Z displacements along each row, column, and plane in the nearest neighbors template, a smaller template can be used with assurance that there will be no close "uncounters". After performing the interaction analysis on the objects within the vector shell, the boundaries of the shell can be checked to ensure that objects on the logical boundary are far enough away spatially. These checks over all the objects in the grid can also be vectorized. If the template has been pared sufficiently to ensure a worthwhile reduction in the number of objects that have to be considered most of the time, there will generally be a fraction for which one or more of the neighbors in the vector shell were not far enough away to ensure that the next neighbor, which is outside the vector shell, can safely be neglected.

For the few objects which may have spatially close neighbors which are removed more than two or three locations logically, a scalar calculation can be performed. It would probably save a factor of two or so in overall computational cost to reduce the vector shells of the nearest neighbors template until the scalar cost competes with the significantly reduced vector cost. The expense of keeping track of X_{max}, Y_{max}, and Z_{max} are minimal, three vector operations per object per near neighbor interaction. The test to determine which few objects require extra (scalar) work is even cheaper, a few vector operations per object. Once an object has been found to require extra work, the scalar search can be extended to whatever logical distance is necessary to ensure that physically nearby objects do not go uncounted.

A 5 × 5 × 5 cubical nearest neighbors template has 62 interactions which will be considered for each object. From empirical evidence to date this is adequate provided the critical radius of consideration is somewhat less than the average separation, here taken to be the original object spacing. Relatively few close uncounters can occur because neglected objects are logically at least 3 and generally 4, 5, or 6 nodes away.

Holes can be added to the MLG, locations which move or stay fixed in space but which don't contain an object. Any object neighboring one of these holes has one fewer real object in its interaction template because of the hole but this obvious disadvantage is balanced by the fact that hole locations can be updated any way necessary to improve the locality and structure of the MLG. By adding or shifting holes about judiciously it may be possible to avoid highly distorted MLGs. The holes would be subject to swapping with objects just as if they were objects but their equations of motion can be different and their interactions with real objects zero.

Figure 4.1 was computed using the 512 particle model with point non-interacting particles and the complete 5 × 5 × 5 interaction template shown in Figure 3.5. The volume around each particle was divided into shells of thickness 1 Angstrom and the number of particles in each radial shell was counted for particles logically outside the 5 × 5 × 5 template to determine how often "close uncounters" occur. A close uncounter occurs when an object gets close physically to another object without coming within the MLG nearest neighbors template and thus ensuring that the interaction is "counted" in the vector sum. Concentrating first on the common features of the two physically identical calculations shown in the figure, we see that probability of an uncounted particle penetrating the interaction volume drops off very rapidly as the distance becomes small and hence the physical interaction would be important. It is one hundred times less likely to find an undetected particle coming within 10 Angstroms than to find one coming within 20 Angstroms. It is another one hundred times less likely to find one coming with 5 Angstroms and no close uncounters were ever found less than about 3 Angstroms.

The problem was repeated with a ninth plane of 64 locations added to the calculation for holes. The holes were given the average location of their six nearest logical neighbors as a propagation law and rapidly mixed throughout the volume with the particles. As can be seen, the probability of unrecorded close encounters is essentially unchanged. In this test, at least, holes don't seem to help much. This is probably because the propagation law used took no detailed account of local grid irregularities signalling a possible "close uncounter" problem. Clearly research is needed to refine this generalization to the point where it appreciably optimizes the MLG representation.

IV.B Fluid Dynamic Applications of the MLG

A few words about the application of the MLG to Lagrangian fluid dynamics is appropriate here. Each node of the grid can be identified with a fluid or vortex element. The advantage is in having a regular grid available to solve the physical evolution equations. Elliptic equations, for example, become amenable to highly efficient, vectorized multigrid methods (DeVore, 1984) on regular $N_x \times N_y \times N_z$ grids even though the fluid elements themselves move randomly. Two-dimensional and four-dimensional problems can be handled just as easily by the same methods.

Work is needed telling how to evaluate spatial derivatives accurately on the distorted MLG. When points are far apart spatially, the fluid cannot be as accurately represented as when they are close. To keep the resolution more nearly uniform than the specific fluid flow may be capable of, it can become necessary to remove nodes where they are crowded and to inject them elsewhere to better resolve some regions. To do this in the MLG involves finding a fluid element which can be merged with a larger one nearby in a manner which conserves mass, momentum, and energy. This frees up a location which can be "shifted" to the correct row, column, and plane to improve a deteriorating local resolution. The process in 2D is quite analogous to the operations needed to order sliding tiles numbered 1 to 15 in the 4 × 4 spaces of a popular child's game. The hole is shifted to the place where it is most needed.

After a shift operation, the same as an ordered series of swaps, it is likely that local swapping may be necessary to reestablish monotonicity. Shifting a line in one direction may well trigger swaps in the other two directions, so the real cost of adding and subtracting Lagrargian nodes locally to control resolution has to be measured (or estimated) for each configuration being considered. Certainly the specific additions, deletions and shifts will have a significant scalar component of computation. It does not make sense to execute a vector swapping iteration over the whole grid unless a significant fraction of the objects are being swapped. However, as long as at least 5% of the objects are being moved, the overall vector swapping iteration probably pays.

For comparison consider another free Lagrangian approach, the Lagrangian Triangular (Tetrahedronal) Grid (Crowley, 1972; Boris and Fritts, 1975; Fritts and Boris, 1979). In this approach the logical grid structure varies in time as the nodes move. The number of nearest neighbors can vary from node to node and the number and identity of these neighbors can vary at a given node as the Lagrangian configurations change. This extra freedom, not allowed in the MLG, is used to maintain a local grid structure optimized to guarantee diagonal dominance of the simplest conservative finite-difference elliptic operator. The price is the loss of local order in the grid and hence no vectorization.

Generalizing this Lagrangian Triangular Grid (LTG) to 3D is straightforward but operationally very complicated. The grid is composed of adaptively restructuring arrangements of tetrahedra in this case. The local grid structure can still be changed as needed to maintain diagonal dominance of the elliptic operator and resolution can be increased or decreased locally as needed. Scalar linked lists become a necessary evil to keep track of nearest neighbors though the resulting algorithms are still of order N.

Clearly the local spatial structure of the MLG is not as "good" as in the generally structured LTG but the global structure compensates for this. The monotonicity conditions specify a meaningful and useful relationship between spatial derivatives and grid differences. As a result, fluid flows with long range correlations, unlike the random particle motions used in earlier tests, may lead to an additional computational expense at specific times. In the smooth flow of large rotating and translating vortices, an initially rectilinear grid might survive many timesteps before any of the local monotonicity constraints are violated. Nearby points would move in almost the same way. Once the fluid rotates far enough, however, monotonicity violations would have to occur. Because of the long range correlation of the motions, a number of swapping iterations may be necessary to reset the MLG.

In a turbulent flow with coherent flow structures at several spatial scales, we can expect intermittant bursts of swapping activity from different scales at different temporal frequencies. Small coherent structures in the flow require fewer swapping iterations to restore monotonicity but will require them more often than large structures when the rotation rates are higher. Though the integrated number of these swapping iterations is large, it is unlikely to be larger than the number of swaps required for random object motion with the same typical distances traversed.

The strong possibility exists that statistics on the number and frequency spectrum of grid swaps may prove to be a very good diagnostic of the progress of turbulent mixing and flow interpenetration. Each swap can be viewed as a single quantum of geometrical upset. The object swapping process is necessarily intermittant and thus presents the possibility of easy integral measures of local intermittancy and turbulence spectrum. Studying the object swapping time series might provide a very direct way of diagnosing such fluid simulations for the onset and character of chaos.

IV.C Other Applications of a Monotonic Logical Grid

The MLG suggests itself for use in multiphase fluid problems. Each grid node could be used to represent a droplet in a spray or a grain of sand in a sandstorm. Droplets could have varying sizes which increase or decrease in time due to local surface effects like condensation, evaporation, or abrasion and all the droplets would not have to be simulated. The accumulation of temporal averages over times and distances short compared to changes in the background flow means that only a small fraction of all the droplets or particles would have to be followed in the MLG to get a good estimate of the interactions of the whole distribution of particles with the background gas.

Collisions of two drops could occasion merging or fragmentation. The MLG can accomodate either by shift operations which transport nodes from where they are no longer needed to new sites where fresh droplets are formed or enter the system. The gaseous background could be represented on a Eulerian mesh to facilitate swapping of mass, momentum and energy back and forth between particles and gas. The volume overlap of MLG cells with cells of the Eulerian grid can be used to circumvent a major complication of Monte Carlo methods, choosing the values of continuum functions at places where there are no particles or Lagrangian nodes. The MLG cells provide a natural way to interpolate back and forth between the two representations.

When insolation of dust or droplet clouds is important, the MLG provides a simple way to assess the radiation opacity along any particular direction. The grid axes can be chosen in a given direction and the swapping algorithm used to resort the points along that direction. There are a number of line-of-sight obscuration problems where this flexibility will be useful.

The Monotonic Logical Grid (MLG) has been given only a geometric context so far. In the applications and examples above, the moving objects are being arranged relative to each other in 3D Cartesian space. Other more abstract applications suggest themselves. The MLG can just as easily represent multidimensional phase spaces for Boltzmann and Vlasov Equations. The grid may also be useful for some classes of problems involving more abstract data organization. The MLG is constructed using a set of monotonicity operators, one for each dimension. These are really relational or comparison operators that return a logical "yes", "maybe", or "no" depending on the results of a generalized comparison between two logical entities. The operators can be black boxes implementing complex, possibly subjective comparison algorithms which need have nothing to do with geometry or physical nearness. The meaning behind the relational operators can be almost anything; for example, "degree of difficulty", "technical merit" and "artistic merit" may be separate awards in the rating of an athletic contest.

V. SUMMARY AND CONCLUSIONS

This report introduced a simple, vectorized algorithm to determine nearest neighbors whose cost scales as the number N of independent objects or locations. This is accomplished by defining a Monotonic Logical Grid (MLG) for storing the object data dynamically so that objects which are adjacent in real space are automatically close neighbors in the logical grid as well. As a simple geometric test problem, a regular $8 \times 8 \times 8$ 3D grid was used to store the position and velocity components of 512 randomly located particles in a cubical domain. For this idealized system the points were given random velocities and the MLG was evolved for many transits of the system by the faster particles. Statistics on near encounters of logically far away points and on the number of restructuring operations required were presented.

It was found that the reconnections of the dynamically changing MLG can generally be computed locally in a very few vectorized iterations without using inefficient gather or scatter operations. Almost all of the grid restructuring occasioned by particles passing each other occurs in the first two or three vectorized iterations. Further optimization is possible by changing to local scalar swapping after a few iterations. It is also found that almost all the spatially closest nodes are nearby in the MLG as well. Two or three logical grid locations effectively defines the spatial near neighborhood except for a vanishing small number of cases which can be detected and corrected inexpensively.

The MLG differs from previous nearest-neighbor algorithms. It effectively removes the constraint of having to associate a cell of the logical grid with a fixed region of real space, but introduces the constraint of only one particle per computational cell. When many of the objects cluster somewhere, a corresponding fraction of the storage locations in the MLG are automatically associated with that region. This means that substantial variations in object density are adaptively gridded by the MLG and large regions of space, as well as computer memory, are not occupied by empty cells.

This algorithm gives regular global orderings of the object data and so allows efficient contiguous vector operations which are longer than the relatively small number of neighbors considered for each object but can be much shorter than the total number N. The algorithm will execute efficiently in small array processors and permits direct partitioning to take advantage of massive asynchronous parallelism in VLSI/VHSIC-based distributed processing systems. The cost to execute the simplest version of the model is one hour on a DEC VAX 11/780 for one thousand particles for 1000 timesteps when a simple force law for the 124 nearest neighbors is used. With a commercially available Distributed Processing System, 5000 interacting objects could be monitored and the data base updated and then restructured thousands of times in about 15 minutes, fast enough for realtime applications.

A number of potential applications were discussed briefly. Obviously other uses will suggest themselves as the good properties and restrictions of the interesting multivalued geometric MLG mapping between real space and relative (logical or computer storage) space becomes better understood. These problem-independent properties will be necessary to the successful application of the representation to practical problems with other mathematical, logical or physical constraints. Swapping and shifting operations and holes were introduced to allow efficient local and global grid readjustments. Practical experience with the MLG is still small, however, so major pathologies may yet be uncovered in some applications.

Many MLG configurations may be possible for the same physical node arrangements and simple examples suggest that the best configurations are much better than the worst. Thus efficient methods of optimizing local and global structure within the monotonicity constraints will eventually be imperative. Additional work is needed on the following questions:

1. What is the mathematical nature of the simple representations for spatial derivative operators and integral conservation operators and how can they be optimized computationally?

2. Is there an algorithm to optimize the grid structure using holes and/or adaptively varied local modifications of the monotonicity functions?

3. What is the cost of not reaching monotonicity every cycle?

4. What is the geometric or information theoretic meaning behind the ambiguity of possible representations, i.e. what kind of an uncertainty principle does this represent?

Acknowledgments

I would like to acknowledge years of informative and rewarding discussions with Dr. Martin Fritts on topics ranging over all aspects of this subject. His diligent and creative efforts on the LTG approach have provided an information base for the development of the MLG algorithm. I would also like to thank Dr. Sam Lambrakos for providing the information in Figures 3.3 and 3.4. This work was supported by the Office of Naval Research projects in Large-Scale Scientific Computing (44-1909, RR014-03-05), Computational Hydrodynamics (44-0573, RR01403-02) and Molecular Dynamics (44-1950-0-5, 61153N), and by the Naval Research Laboratory.

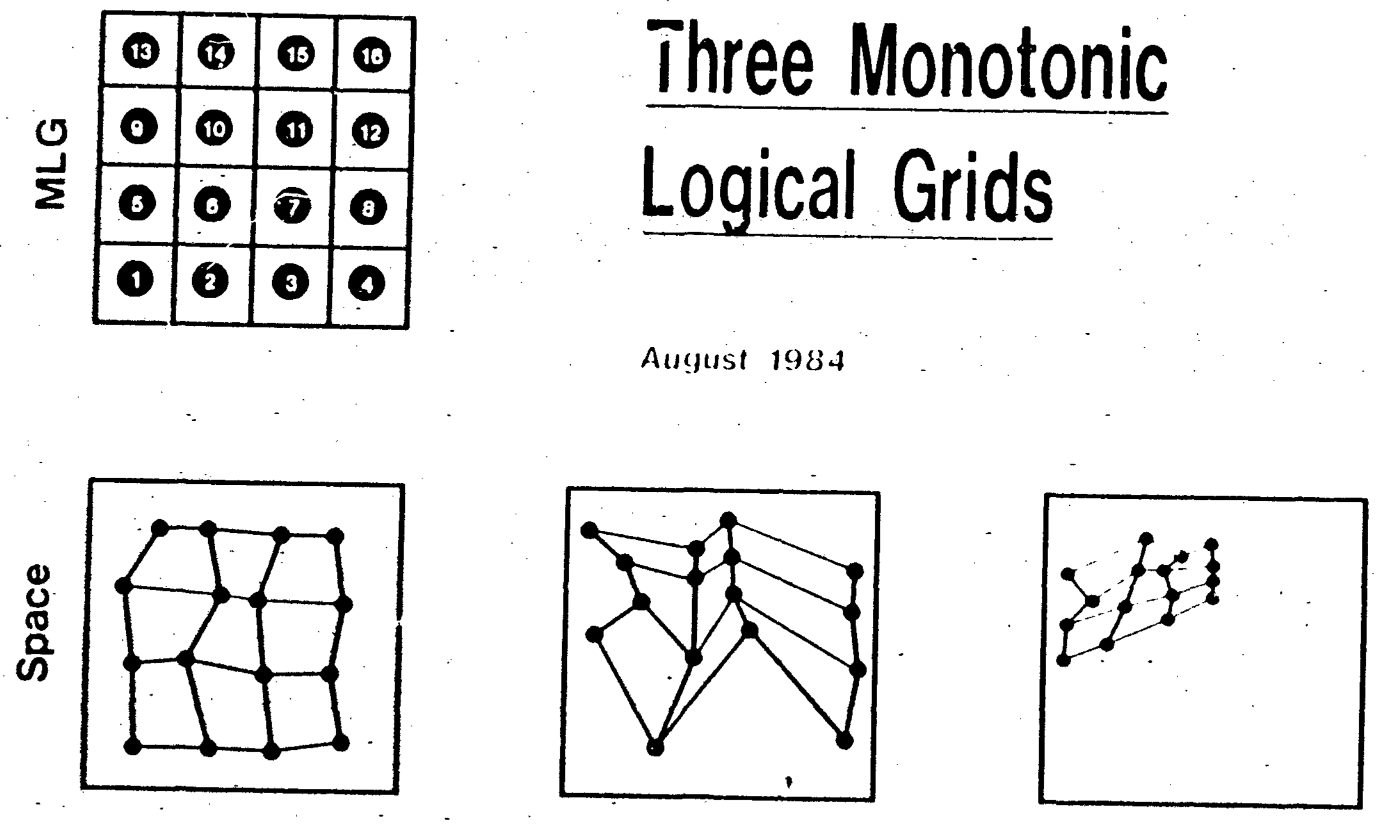
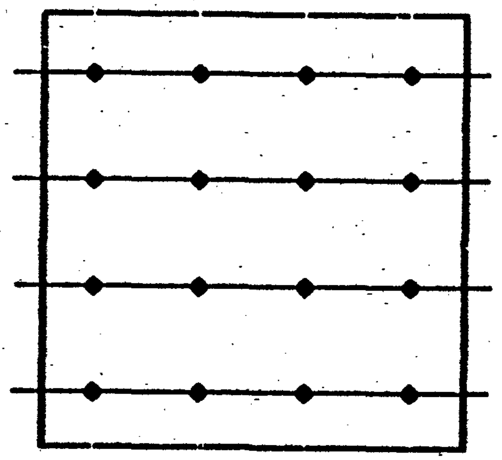


Figure 3.1 Three Different Spatial Configurations and the Corresponding Monotonic Logical Grid



Three Monotonic Logical Grids from Identical Data

August 1984

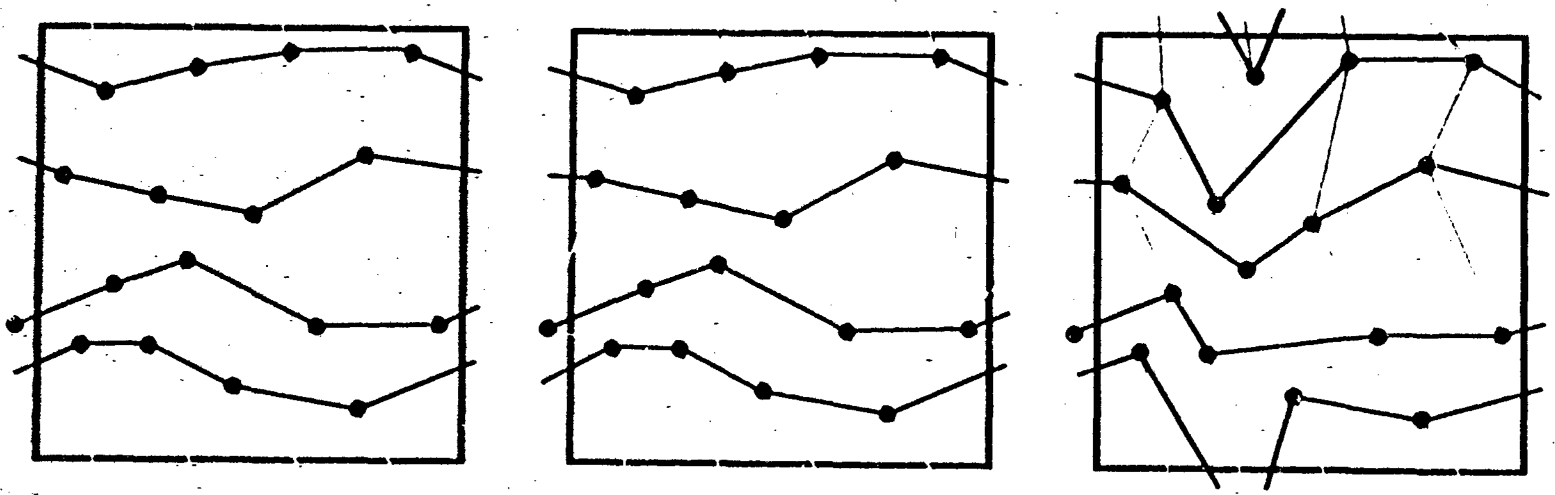


Figure 3.2 Three Monotonic Logical Grids from Identical Data

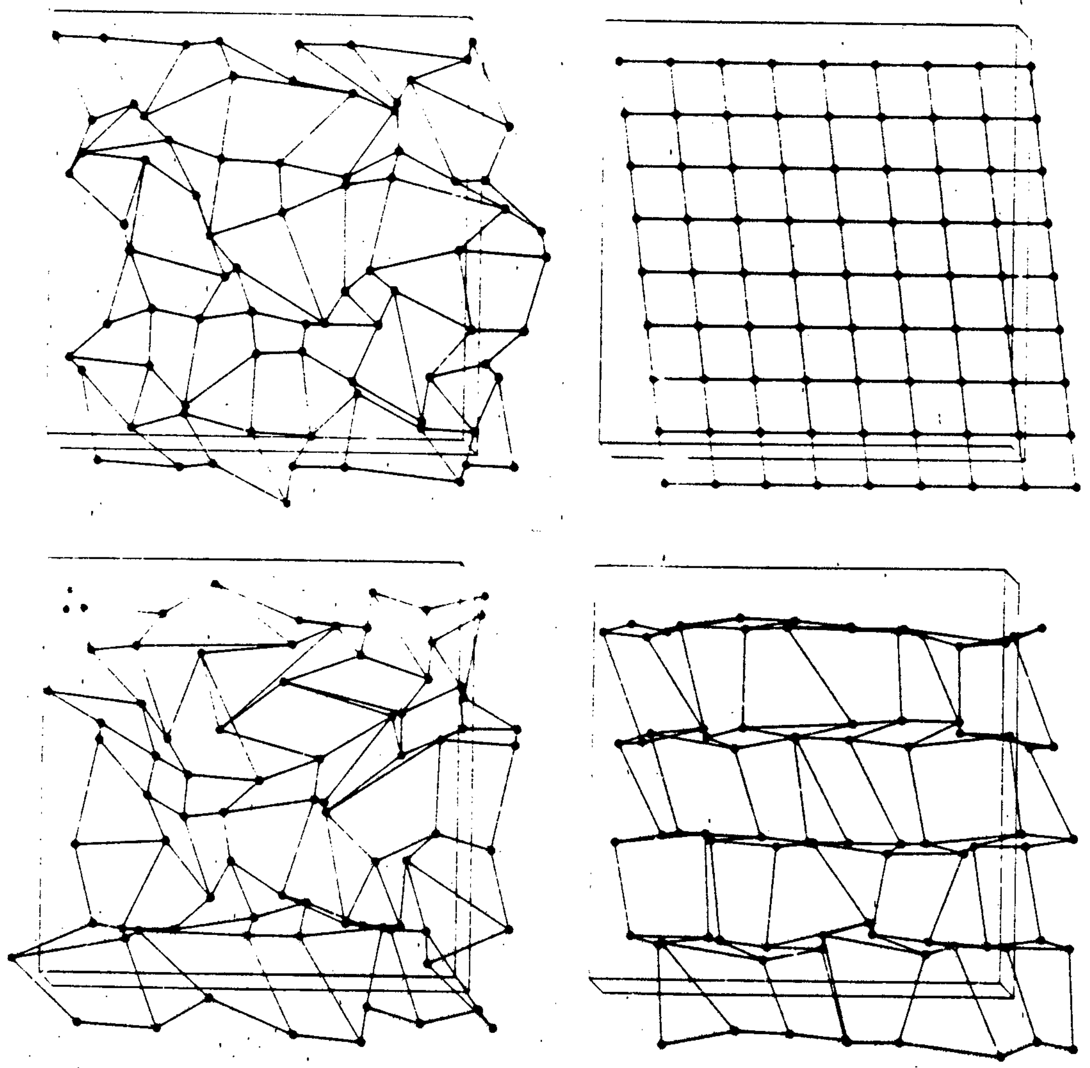


Figure 3.3 The MLG for One Plane of the 512 Particle Test Problem

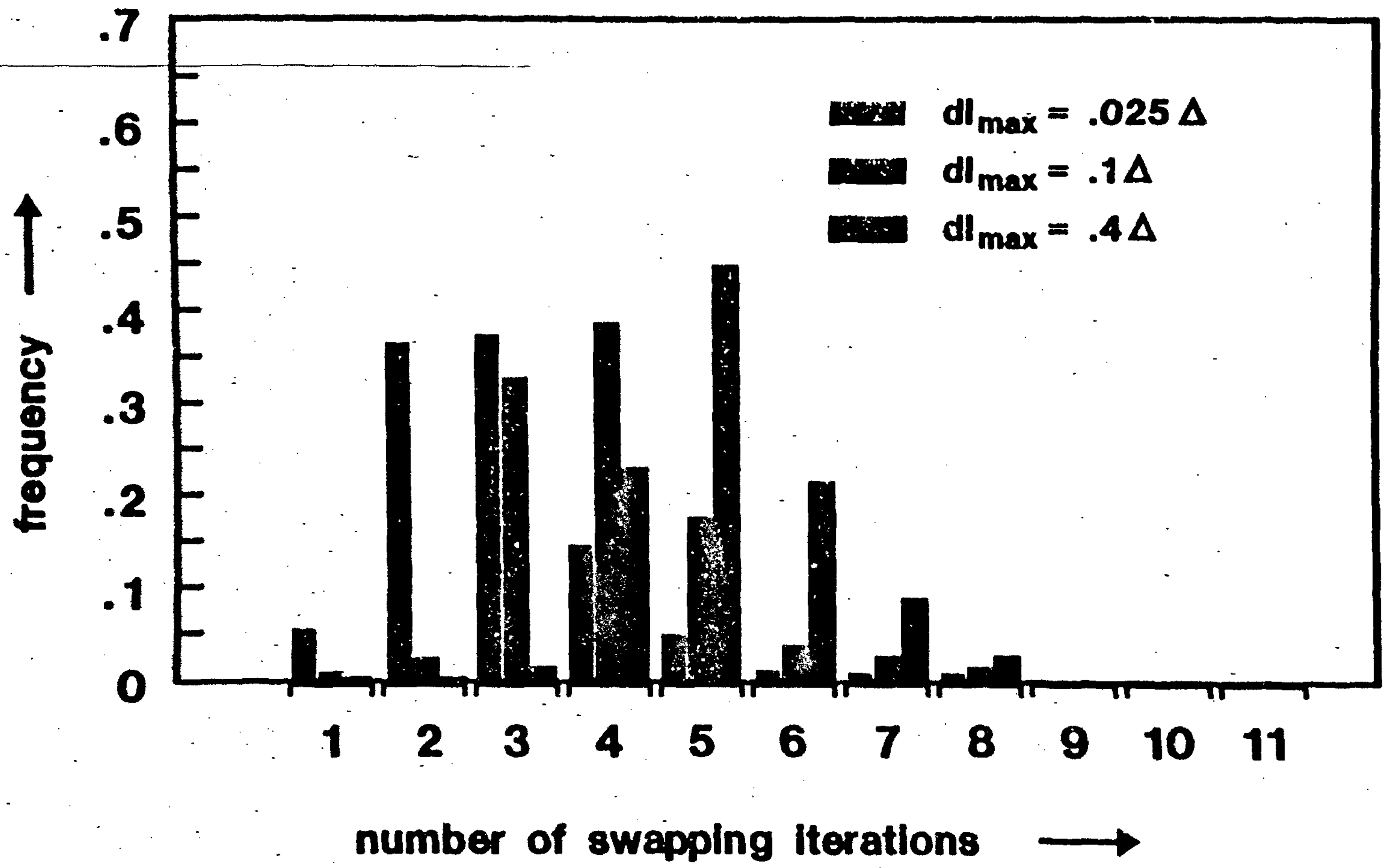
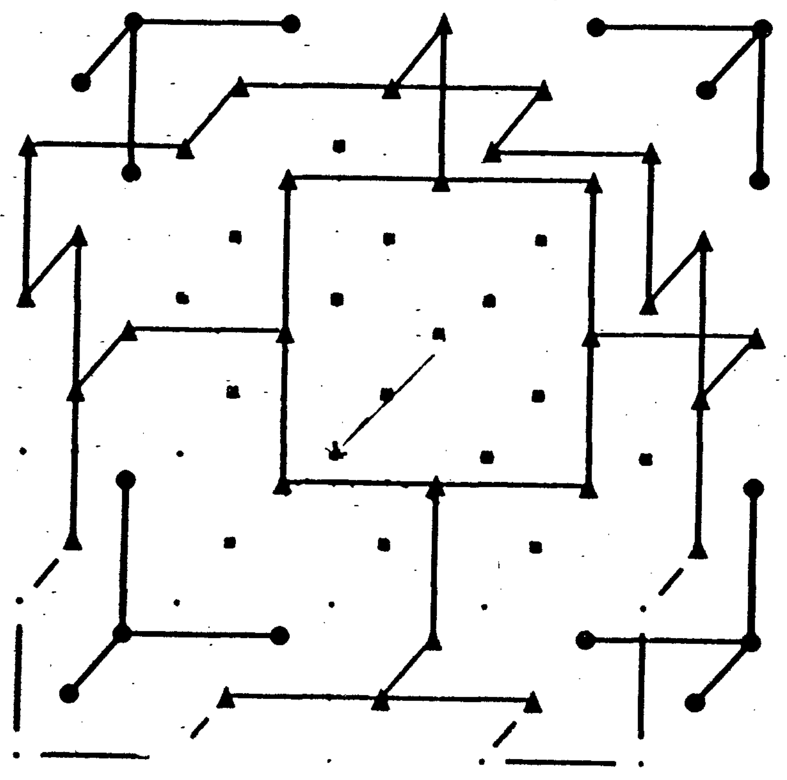


Figure 3.4 Frequency of Requiring N Swapping Iterations to Construct a Monotonic Logical Grid

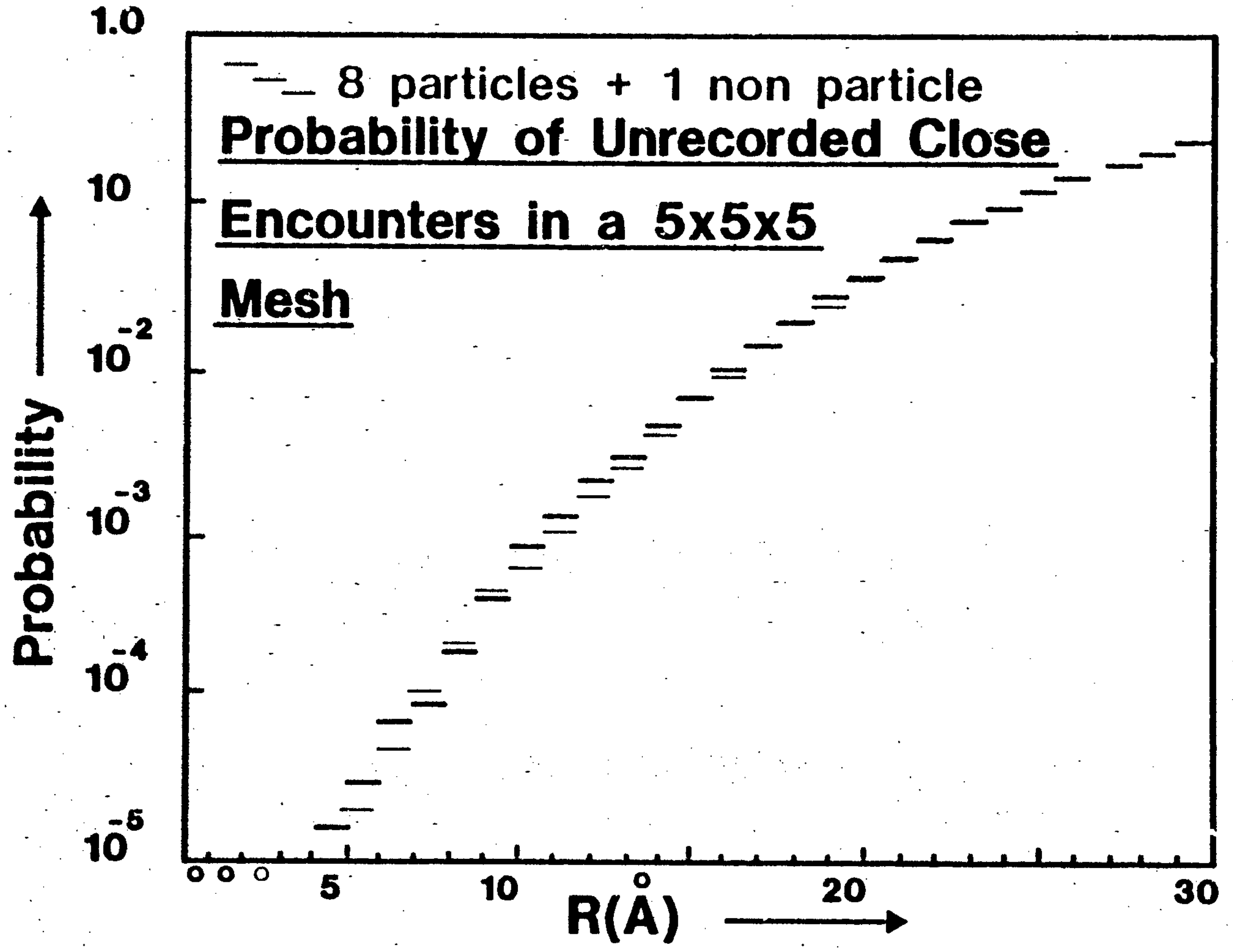
LOGICAL DISPLACEMENTS OF NEAREST NEIGHBORS



■ 16 pts
▲ 30 pts
● 16 pts

August 1984

Figure 3.5 Logical Displacements of Nearest Neighbors



August 1984

Figure 4.1 Probability of Unrecorded Close Encounters in a 5 × 5 × 5 Mesh

REFERENCES

1. R.W. Hockney and J.W. Eastwood, "Computer Simulation Using Particles", Chapter 8, pp267-304 (McGraw-Hill Inc, New York, 1981).

2. W.F. van Gunsteren, H.J.C. Berendsen, F. Colonna, D. Perahia, J.P. Hollenberg, and D. Lellouch, "On Searching Neighbors in Computer Simulations of Macromolecular Systems", Journal of Computational Chemistry, Vol. 5, No. 3, pp272-279 (1984).

3. G.C. Fox and S.W. Otto, "Algorithms for Concurrent Processors", Physics Today, pp50-59, May 1984.

4. L.M. Adams and H.F. Jordan, "Is SOR Color-Blind?", ICASE Report No. 84-14, NASA Langley Research Center, May 1984.

5. J.P. Boris, K.L. Hain, and M. Fritts, "Free Surface Hydrodynamics Using a Lagrangian Triangular Mesh", Proceedings of the First International Conference on Numerical Ship Hydrodynamics, Gaithersburg, MD, October 20-22, p. 683, 1975.

6. M.J. Fritts and J.P. Boris, "The Lagrangian Solution of Transient Problems in Hydrodynamics Using a Triangular Mesh", J. Comp. Phys., Vol 31, p173, May 1979.

7. W.P. Crowley, "FLAG: A Free Lagrange Method for Numerically Simulating Hydrodynamic Flows in Two Dimensions", in Proceedings of the Second

International Conference on Numerical Methods in Fluid Dynamics, (Springer-Verlag, New York, p. 37, 1971).

8. C.R. Devore, "Vectorization and Implementation of an Efficient Multigrid Algorithm for the Solution of Elliptic Partial Differential Equations", NRL Memorandum Report 5504, January, 1985.